PATENT ABSTRACTS OF JAPAN

(11)Publication number : 10-283192
(43)Date of publication of application : 23.10.1998

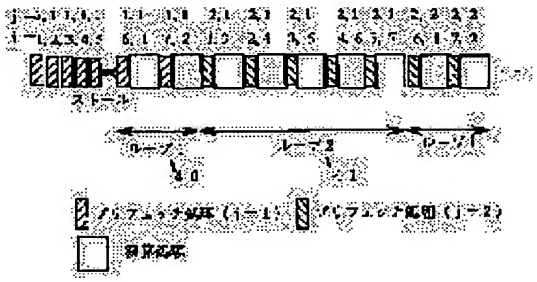
(51)Int.Cl. G06F 9/45

(21)Application number : 09-090470
(22)Date of filing : 09.04.1997

(71)Applicant : HITACHI LTD
(72)Inventor : TANAKA GIICHI
ITO SHINICHI
TSUSHIMA YUJI

(54) PREFETCHING CODE GENERATION SYSTEM

(57)Abstract:
PROBLEM TO BE SOLVED: To hide a main memory penalty in the case that a data area accessed by a loop exceeds a cache capacity by dividing an inner side loop into two and starting the prefetching of data in a second half loop at the time of the repetition of an outer side loop one before for the data required in a first half loop and in the first half loop for the data required in the second half loop.
SOLUTION: The inner side loop is divided into two and a code for incorporating the prefetching processing of the data required at the first part of the repetition processing of the outer side loop one after in the second half loop 41 and incorporating the prefetching of the data required in a second half in the repetition processing of the same outer side loop in the first half loop 40 is generated. For the distribution of the front half loop 40 and the second half loop 41, the processing time of the second half loop 41 is turned to be more than main memory penalty time. In the case that the processing time of the second half loop 41 can not be turned to be more than the main memory penalty, loop development is performed relating to the outer side loop and an arithmetic amount is increased so as to completely hide the main memory penalty.



LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's decision of rejection]

[Date of extinction of right]

(19)日本国特許庁（ＪＰ）　(12)公開特許公報（Ａ）　(11)特許出願公開番号

特開平10－283192

(43)公開日　平成10年(1998)10月23日

(51)Int.Cl.[6]	識別記号	Ｆ１	
Ｇ０６Ｆ　9/45		Ｇ０６Ｆ　9/44	３２２Ｇ

審査請求　未請求　請求項の数５　ＯＬ　（全 10 頁）

(21)出願番号　特願平9－90470

(22)出願日　平成９年(1997)４月９日

(71)出願人　000005108
株式会社日立製作所
東京都千代田区神田駿河台四丁目６番地

(72)発明者　田中　義一
東京都国分寺市東恋ケ窪一丁目280番地
株式会社日立製作所中央研究所内

(72)発明者　伊藤　信一
神奈川県横浜市戸塚区戸塚町5030番地　株式会社日立製作所ソフトウェア開発本部内

(72)発明者　對馬　雄次
東京都国分寺市東恋ケ窪一丁目280番地
株式会社日立製作所中央研究所内

(74)代理人　弁理士　小川　勝男

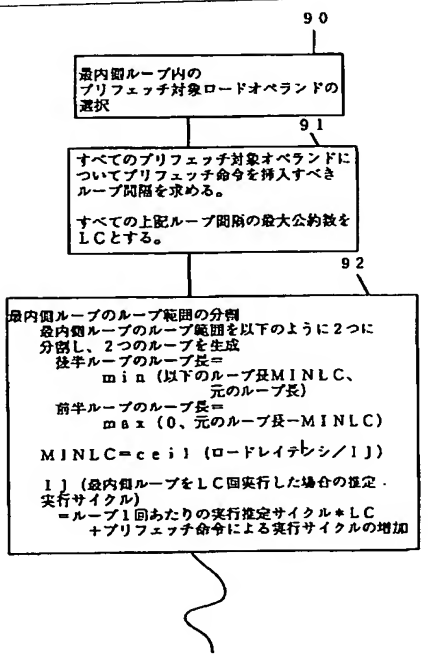
(54)【発明の名称】　プリフェッチコード生成方式

(57)【要約】

【課題】多重ループにおいて、ループでアクセスするデータ領域がキャッシュ容量を超えた場合に、主記憶ペナルティを隠蔽するオブジェクト生成方式を提供する。

【解決手段】内側ループ処理を２つにわけ、後半ループで、１つ先の外側ループの繰り返しの処理の最初の方で必要となるデータのプリフェッチ処理を組み込み、前半ループで、同一外側ループの繰り返し処理で、後半で必要となるデータのプリフェッチ処理を組み込むコードを生成することで達成される。ここで、ループの分配は後半ループの処理時間が主記憶ペナルティ時間以上となるように行う。

図９



【特許請求の範囲】

【請求項１】ソースプログラムをオブジェクトプログラムにコンパイルする方式において、上記ソースプログラムの多重ループ部分に対して、上記多重ループのそれぞれがループの実行の直前にはループ実行回数が決まるループの場合、上記多重ループのうち外側ループをループ１，最内側ループをループ２とするとき、上記ループ２内にある演算で必要となるデータの一部を、上記ループ１に関して前のループ繰り返し処理時に、キャッシュに転送するプリフェッチコードを生成することを特徴とするコード生成方式。

【請求項２】ソースプログラムをオブジェクトプログラムにコンパイルする方式において、上記ソースプログラムの多重ループ部分に対して、上記多重ループのそれぞれがループの実行の直前にはループ実行回数が決まるループの場合、上記多重ループのうち外側ループをループ１，最内側ループをループ２とするとき、上記ループ２を２つのループ繰り返し範囲に分割し、それぞれを前半ループ，後半ループとするとき、上記後半ループのコードは、上記ループ１に関して次の繰り返し処理時に上記前半ループの演算の最初に必要となるデータをキャッシュに転送するプリフェッチコードともともとのコードからなり、上記前半ループのコードは、上記前半ループの演算で必要となるデータのうち、上記後半ループでキャッシュに転送されるデータ以外のデータをキャッシュに転送するプリフェッチコードともともとのコードとからなることを特徴とするコード生成方式。

【請求項３】請求項２のコード生成方式において、上記ループ２のループ繰り返し範囲の分割において、上記後半ループの繰り返し数は、上記後半ループの処理時間を推定し、上記処理時間がメモリからキャッシュへの転送時間である主記憶アクセスペナルティを超えるように求めた推定ループ繰り返し数か、もともとの上記ループ２のループ繰り返し数のうち小さい繰り返し数を設定することを特徴とするコード生成方式。

【請求項４】請求項３のコード生成方式にて、上記後半ループの処理時間を推定し、上記処理時間がメモリからキャッシュへの転送時間である主記憶ペナルティを超える推定ループ繰り返し数が、ある与えられた基準値より、大きい時は、ループ１に関してループ展開ができないか解析を行い、展開可能な場合、展開後の上記後半ループの処理時間を推定し、上記処理時間がメモリからキャッシュへの転送時間である主記憶アクセスペナルティを超える推定ループ繰り返し数が、上記基準値より小さくなるまでループ１に関するループ展開を行ったコードを生成することを特徴とするコード生成方式。

【請求項５】請求項２のコード生成方式にて、上記プリフェッチが必要なループ繰り返し間隔数を求め、上記後半ループのコードは、上記ループ１に関して次の繰り返し処理時に上記前半ループの演算の最初に必要となるデータをキャッシュに転送するプリフェッチコードともともとの処理コードを上記展開数分だけ複製したものからなり、上記前半ループのコードは、上記前半ループの演算で必要となるデータのうち、上記後半ループでキャッシュに転送されるデータ以外のデータをキャッシュに転送するプリフェッチコードともともとの処理コードを上記展開数分だけ複製したコードからなることを特徴とするコード生成方式。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】本発明は、命令レベル並列処理を行うプログラムの実行方式に係わり、プログラムがアクセスするデータが大きく、キャッシュに入りきれない場合のループに好適なコード生成方式に関する。

【０００２】

【従来の技術】スーパコンピュータの１つの方向として、スカラプロセッサをノードプロセッサとする並列処理方式が有望視されている。スカラプロセッサを用いた並列スーパコンピュータが期待されるのは、半導体技術の進歩によるクロック周波数の向上、複数の並列実行可能な演算器を有効に生かすスーパスカラ方式等の命令レベル並列処理の実現により、スカラプロセッサの処理性能が飛躍的に向上しているためである。しかしながら、その高いスカラプロセッサ処理能力は、キャッシュが有効に働くときにのみ達成される。

【０００３】スカラプロセッサは、一般的に命令処理装置とキャッシュ及びメモリ装置を有する。スカラプロセッサはメモリにプログラムとデータを格納し、プログラムに記述された命令に従いメモリ中のデータを処理する。キャッシュは命令処理装置からの参照時間の短い比較的小容量の記憶手段であり、プログラム及びデータの一部を一時的に格納する。命令実行にあたり必要なデータはメモリから読み出されるが、同時にデータを含むデータブロックはキャッシュを構成するラインにコピーされ、以後当該ブロック内のデータに対する参照が指定されたときは上記キャッシュのラインからデータを参照する。このメモリからキャッシュラインへのデータブロックの転送をライン転送と呼ぶ。

【０００４】命令実行にあたり必要データがキャッシュにない場合、これをキャッシュミスと呼ぶが、キャッシュミスが発生するとライン転送が実行される。従来のスカラプロセッサでは命令実行に伴いこのライン転送が発生すると、その完了まで当該命令の実行が待ち合わされる。従って、キャッシュミスが多発するとライン転送に伴う待ち合わせによりプログラムの実行時間が増大し、スカラプロセッサの処理性能が劣化するという問題があった。特に、大規模科学技術計算では、データ領域が大きく、データの局所性が少ないという性質があるためキャッシュミスによる性能低下は深刻であった。

【０００５】これに対し、最近では予めプログラムにお

いて、キャッシュミスを引き起こす可能性のある命令に先だってデータの先読みを指示する特殊な命令を実行させることで上記ライン転送に伴う性能劣化を回避する試みがなされている。このデータ先読みをプリフェッチと呼ぶ。例えば、米国ＩＢＭ社のマイクロプロセッサPowerＰＣ では指定したオペランドアドレスに位置するデータをキャッシュに読み込むプリフェッチ命令がある。この動作において、キャッシュミスを発生した場合、プリフェッチ命令の完了を待ち合わせることなくライン転送を行う。従って上記データを参照する命令を実行する時には上記データがキャッシュに入っているために上記性能劣化が回避される。

【０００６】このようなプリフェッチ命令を利用して、データ転送と演算をオーバラップさせることで、実質的に主記憶レイテンシを隠蔽させるコード生成技術が論文ToddC.Mowry, Monica S.Lam, Anoop Gupta「Design and Evaluation of a CompilerAlgorithm for Prefetching」ASPLOS-V, ACM 0-89791-535-6/92/0010/0062 pp.62-73に示されている。

【０００７】

【発明が解決しようとする課題】しかしながら、上記の従来のコード生成技術では以下に記すような問題があった。図２に示すようなＤＯ２０（２０）及びＤＯ１０（２１）で示される多重ループに対するプリフェッチコードの生成において、内側ループＤＯ１０（２１）のみしか考慮されていなかった。このことを例を用いて具体的に問題点を示す。

【０００８】まず、以下のコード生成で前提としたアーキテクチャの仮定を述べる。対象とするスカラプロセッサはロード／ストア命令が２個，浮動小数点演算が２個、及び整数演算が２個、同時に実行でき、キャシュラインサイズが３２バイト，キャッシュミス時の主記憶ペナルティを７５サイクルと仮定する。

【０００９】図２のソースプログラムに対して、コンパイラは、まず、ローカリティ解析などによりプリフェッチ対象オペランドを探す。その結果、参照ｂ（ｉ，ｊ）２２がプリフェッチ対象オペランドであるとする。オペランドの大きさを８バイトとすると、１回のプリフェッチで、対象とするオペランドを含む連続した３２バイトのデータがキャッシュに格納されるため、ループ繰り返しごとにプリフェッチ命令を実行していたのでは無駄で、この例の場合、４回に１回の割合でプリフェッチ命令を出力すればよい。このため、一般的には、コンパイラのループ構造変換部で、最内側ループ２１に関して４倍のループ展開を行う。

【００１０】この結果を図３に示す。ここで、ループ２１がループ２５とループ２６に分割されている。ループ２６は、ループ展開の余りループである。以後、簡単のため、ループ展開の余りループに関しては省略することにする。そして、通常の最適化、コード生成の後に、プリフェッチコードの生成を行う。

【００１１】プリフェッチコードの生成は以下のように行う。まず、ループ２５のループ１回あたりの実行サイクル数を推定する。ループ２５では、ロード命令が４個，ストア命令４が４個，浮動小数点演算が４で、プリフェッチ命令が１であるので、ループ１回あたりの処理サイクル数は５サイクル（＝max（ceil（（４＋４＋１）／２），２／２）と推定できる。ここで、ceil関数は、引数の値を超えない最小の整数を意味する。キャッシュミス時の主記憶ペナルティは７５サイクルであるので、ループ処理では、１５回（ceil（７５／５）＝１５）後で使用されるデータのプリフェッチを行うコードを生成すると、主記憶ペナルティを隠蔽することができる。

【００１２】図４は、図３のコードにプリフェッチ命令を付加したものである。ここでＡ部３０は、ループ繰り返しの最初の方で必要となるデータのプリフェッチを行う部分，Ｂ部３１は演算処理とループ繰り返しに関して１５回後で使用するデータのプリフェッチ命令を含むループ部分、Ｃ部３２はループ処理の後半部でプリフェッチ命令を含まないループ部分である。

【００１３】図５はその時の実行の様子を模式的に示したものである。横軸が時間，斜線箱がプリフェッチ処理，空箱が演算処理である。ただし、図面の都合上、図４のプログラムとは数値的に一致はしていない。つまり、図４のＡ部は１５回のプリフェッチループであるが、図５では、３回に省略されている。ここで、問題であるのは、Ａ部であり、主記憶アクセスペナルティが顕在化する。これは、特にループ処理回数が小さい場合に問題となる。

【００１４】以下では、このコードの性能をみつもる。ループ長がＮ＝１００の場合、Ａ部の実行時間は、少なくとも主記憶アクセスペナルティの７５サイクル、Ｂ部は５０サイクル（ループ１回あたり５サイクル×ループ１０回実行）、そしてＣ部は６０サイクル（ループ１回あたり４サイクル×ループ１５回実行）で計１８５サイクルで、全体の時間の内４０％（７５／１８５×１００％）が主記憶待ちのオーバヘッドとなっている。さらに、ループ長が５０と小さい場合には、Ａ部は少なくとも７５サイクル，Ｂ部はループ長が小さいため実行されず０サイクル，Ｃ部は、４８サイクル（ループ１回あたり４サイクル×ループ１２回実行）で、計１３３サイクルで、全体の時間の内５６％（７５／１３３×１００％）が主記憶待ちのオーバヘッドとなる大きな問題がある。Ａ部で、少なくとも７５サイクルと書いたのは、現実の計算機では、同時に処理できるプリフェッチ命令の数には限界があるためである。ここでは、十分大きいと仮定した。

【００１５】また、上記で説明したプリフェッチ命令の削減のための内側ループ展開法を用いたコード生成は、以下の問題がある。通常、このようなループの構造変換

はコンパイラの前半部分で行われ、その後、最適化処理が施され、コード生成部でプリフェッチ命令が挿入されると考えられる。こうすると、コンパイラ処理の大半で、ループ展開による大きな中間語を扱わねばならず、長大なコンパイル時間が必要になる。ここでは、ラインサイズ３２バイトでの例を示したため、高々４倍の展開であったが、ラインサイズ１２８バイトの計算機では１６倍展開ものループ展開が必要となり、さらに深刻となる。

【００１６】本発明の目的は、上記の問題点を解決するプリフェッチ命令を用いたコード生成方式を提供することにある。

【００１７】

【課題を解決するための手段】上記の目的は、図６に示すように、内側ループ処理を２つにわけ、後半ループ４１で、１つ先の外側ループの繰り返しの処理の最初の方で必要となるデータのプリフェッチ処理を組み込み、前半ループ４０で、同一外側ループの繰り返し処理で、後半で必要となるデータのプリフェッチ処理を組み込むコードを生成することで達成される。

【００１８】例では、外側ループＪ＝１の、内側ループＩ＝３，４，５，６，７の処理の際に、外側ループＪ＝２の内側ループＩ＝１，２，３，４，５で必要となるデータをプリフェッチする命令を挿入し、外側ループＪ＝１の、内側ループＩ＝１，２の処理の際に、内側ループの後半Ｉ＝６，７で必要となるデータをプリフェッチする命令を挿入する。ここで、前半ループ４０と、後半ループ４１の分配は、後半ループ４１の処理時間が主記憶ペナルティ時間以上となるようにループ処理を分割する。

【００１９】ここで、内側ループの演算数が小さくループ長が短い場合は、後半ループ４１の処理時間を主記憶ペナルティ以上にできない場合がある。この場合、当該ループの処理の時間が、想定した規定値以上のループ長の場合には、完全に主記憶ペナルティを隠蔽できるように、外側ループに関してループ展開を行い、演算量を増加させて、この目的を達成する。

【００２０】また、プリフェッチ命令の削減のための内側ループ展開法を用いたコード生成によるコンパイル時間の増大の問題は、コンパイラ処理の後半のコード生成処理において、コードを複製する方式により解決できる。

【００２１】

【発明の実施の形態】以下、本発明のコンパイラにおける実施例を図を参照しつつ説明する。図１にコンパイラ全体の構造を示す。図１のソースプログラム１が、構文解析２によって中間語３に変換される。ループ構造変換部４は、これを入力として、多重ループに関するプリフェッチコードの主記憶レイテンシの隠蔽される割合を高くするために外側ループに関するループ展開を行い、中間語５を出力する。最適化部６は、通常の公知の伝統的な最適化を行い、中間語７を出力する。コード生成部８は、プリフェッチ命令を含むオブジェクトコードを９を、中間語７をもとに生成する。本発明は４及び８に係わり、オブジェクトコード９の実行効率を向上させるものである。

【００２２】図１のループ構造変換部４のうち、多重ループにおける主記憶アクセスペナルティの効率的な隠蔽に関わる部分を図７に示す。図７に入力するソースプログラム１として、図２のFORTRAN プログラムを例としてあげ説明する。図２のDO20（２０），ＤＯ１０（２１）はループを示し、これらは２重ループを構成している。このようなプログラムに対して図７は以下のような処理を行い、中間語５に変換する。

【００２３】図７の処理は、ソースプログラム中の多重ループ５０を順次処理する。判定５１は、多重ループを考慮したプリフェッチコードを出力する多重ループを選択する。ここでは、１つのループの中に、１つのループを含む正規型多重ループに限定する。図２のソースプログラムでは、外側ループＤＯ２０（２０）の中に、１つのループＤＯ１０（２１）しかない構造であるので適合する。この例で、外側ループＤＯ２０（２０）の中に、複数のＤＯループがある場合は、本発明では、多重ループを考慮したプリフェッチ命令は出力しないので、ループ構造が適合しないと判断する。

【００２４】処理５２では、元々の内側ループ１回あたりの実行サイクル数を推定する。ループ内には、浮動小数点演算が１個，ロード演算が１個，ストア演算が１個であるので、仮定しているアーキテクチャでは、ループ１回あたり、ＩＩ＝max(ceil（（１＋１）／２），ceil（１／２））＝１サイクルである。

【００２５】処理５３では、次の外側ループでの処理に必要なデータの主記憶アクセスペナルティを完全に隠蔽する最小のループ長ＭＩＮＬを求める。この場合、主記憶アクセスペナルティは７５であるので、ＭＩＮＬ＝７５となる。つまり、このままでは、内側ループ長が７５未満であると、次の外側ループの実行に必要なデータをオーバヘッドなしに準備できなく、データ待ちの状態が発生することになる。実際のプログラムにおいては、このループ長よりも小さい場合も多い。

【００２６】そこで、仮にコンパイラのコード生成の方針として、ループ長４０以上であれば主記憶アクセスペナルティを完全に隠蔽できるコードを生成することにする。判定５４において、規定値とは、このようにコンパイラのコード生成方針で決めた値である。図２のソースプログラムの場合、ＭＩＮＬ＝７５，規定値＝４０であるので、判定５５に進む。判定５５は、外側ループ展開を行っても、もとのプログラムと同一の処理となるか、データ依存関係からの検査を行う。外側ループ展開が可能か否かは、当該多重ループのループ交換が可能か否か

と同一であり、これは公知技術、例えば田中義一，岩沢京子「ベクトル計算機のためのコンパイル技術」情報処理，Vol.31,No.6,pp.736－743(1990) に記述してある方式を用いればよい。図２のソースプログラムではＤＯ２０（２０）とＤＯ１０（２１）のループを交換しても同一の結果を得ることができるので、外側ループ展開が可能である。

【００２７】処理５６では、外側ループの展開数を決めて、外側ループ展開を中間語上で行う。図の例ではceil（７５／４０）＝２倍展開を行い、図８のような結果を得る。ここでは、中間語として、フォートランライクに表現したものを用いている。ＤＯ２０（６０）とＤＯ１０（６１）が、外側ループに関して２倍展開した多重ループ、ＤＯ１１（６２）は、外側ループに関して２倍展開した余りループである。以後、余りループに関しては簡単のため、説明は省略する。判定５７により、すべての多重ループに対して上記の処理を行う。

【００２８】その後図１において、通常の中間語に対する最適化部６の処理を行い、コード生成８を行う。コード生成部８において、多重ループにおける主記憶アクセスペナルティの効率的な隠蔽に関わる部分を図９，図１０に示す。

【００２９】処理９０は、最内側ループ内のプリフェッチ対象ロードオペランドの選択を行う。選択の方法は、例えば、公知技術である前記文献ACM 0-89791-535-6/92/0010/0062 pp.62-73に従う。この結果、図８のＤＯ１０内のｂ（ｉ，ｊ）とｂ（ｉ，ｊ＋１）がプリフェッチ対象オペランドとなるとする。

【００３０】処理９１は、すべてのプリフェッチ対象オペランドについてプリフェッチ命令を挿入すべきループ間隔を求め、それらの最大公約数を求める。キャッシュラインサイズが３２バイトであるため、ｂ（ｉ，ｊ）をプリフェッチすると、ｂ（ｉ，ｊ）を含む連続３２バイトのデータがプリフェッチされる。仮に、ｂ(ｉ，ｊ)がキャッシュラインの先頭にあれば、ｂ（ｉ，ｊ），ｂ（ｉ＋１，ｊ），ｂ（ｉ＋２，ｊ），ｂ（ｉ＋３，ｊ）のデータプリフェッチされるので、ループ４回に１回の割合でプリフェッチ命令を発行すればよい。同様に、ｂ（ｉ，ｊ＋１）もループ４回に１回でよい。ｂ（ｉ，ｊ）及びｂ（ｉ，ｊ＋１）に対するプリフェッチ命令を挿入すべき間隔はそれぞれ４であるため、全体で、プリフェッチ命令を発行するループ間隔は上記の最大公約数であるＬＣ＝４となる。

【００３１】処理９２では、最内側ループのループ範囲を２つに分割する。この処理を記述した箱のなかに記すように、ループ後半のループ長は、実行時間を主記憶アクセスペナルティ以上にするループ長か、または、ループ長が短く、主記憶アクセスペナルティ以上の実行時間にならないときはもともとのループ長を設定し、ループ前半の処理は残りとする。従って、前半ループは実行されないこともある。

【００３２】図８のＤＯ１０（６１）のループ１回あたりの実行推定サイクルは２（＝max（ceil（（２＋２）／２），ceil（２／２））），ＬＣ＝４，プリフェッチ命令はｂ（ｉ，ｊ）とｂ（ｉ，ｊ＋１）に対して出るので、プリフェッチ命令による実行サイクルは１である。このため、最内側ループをＬＣ回実行した場合の推定実行サイクルは９サイクルとなる。このため、後半ループのループ長は、min(元々のループ長、ceil（７５／９））となる。その結果、図８のＤＯ１０（６１）は、図１１の前半ループＤＯ１０（６５），後半ループＤＯ１１（６６）に分割できる。

【００３３】図１１では、前半ループＤＯ１０（６５），後半ループＤＯ１１（６６）を、ＤＯループの構造で示してある。このままであると、実際のオブジェクトコードとのレベルギャップがあるので、最内側ループをマシン語レベルに近くした中間語表現を図１２に示す。ここで、文７０－文７８が図１１での前半ループDO10（６５）に対応し、文８０－８８が図１１での後半ループＤＯ１１（６６）に対応する。

【００３４】ここで、文７１，文７２，文７３，文７６，文７７，文７８でＤＯループ１０の処理を意味する。変数ｃｎｔに処理すべきループ処理長を７１で設定し、文７６で１つ減じ、文７７で繰り返す。文７４はもともとのループ本体での処理、文７０はループ制御変数の初期設定で、文７５がループ制御変数の更新を意味する。

【００３５】同様に、文８１，文８２，文８３，文８６，文８７，文８８でＤＯループ１１の処理を意味する。変数cnt に処理すべきループ処理長を８１で設定し、文８６で１つ減じ、文８７で繰り返す。文８４はもともとのループ本体での処理、文８０はループ制御変数の初期設定で、文８５がループ制御変数の更新を意味する。

【００３６】図１０に進んで、処理９３は、前半ループに対するコード生成を行う。まず、内側ループ後半のためのプリフェッチ命令の挿入を行う。図１３の文１００，文１０１，文１０２がこのコードである。文１００により、内側ループでceil（７５／９）回先に必要となるデータのアドレスを計算する。文１０１がプリフェッチ命令で、文１０２が、ＬＣ回おきにプリフェッチコードを出すためのアドレス更新コードである。次にループボディコードを３（＝ＬＣ－１）回複製して、挿入しする。図１２でのループボディ文７４－７７を複製し、文１０３－文１０６，文１０７－文１１０となる。ループ終了判定コード文７７，文１０６，文１１０の飛び先のlab1 (文７３）から、ループ直後のlab2 (文７８）に変更する。そして、再度、ループボディのコードを複製し、文１１１－１１４に挿入する。

【００３７】処理９４は、後半ループに対するコード生

成を行う。前半ループとの違いは、前半ループでは、プリフェッチ対象オペランドの先頭アドレスが、当該外側ループでの内側ループ処理の後半で必要になるデータに対し、１つ先の外側ループの内側ループ処理前半で必要となるデータである点である。最初に、次の外側ループで必要になるデータのためのプリフェッチ命令の挿入を行う。図１４の文120,文１２１，文１２２がこのコードである。文１２０により、次の外側ループで必要となるデータのアドレスを計算する。文１２１がプリフェッチ命令で、文122が、ＬＣ回おきにプリフェッチコードを出すためのアドレス更新コードである。次にループボディコードを３(＝ＬＣ－１)回複製して、挿入する。図１２でのループボディ文８４－８７を複製し、文１２３－文１２６，文１２７－文１３０となる。ループ終了判定コード文８７，文１２６，文１３０の飛び先のlab3（文８３）から、ループ直後のlab4（文８８）に変更する。そして、再度、ループボディのコードを複製し、文１３１－１３４に挿入する。

【００３８】処理９５は、多重ループの入り口に、初期プリフェッチ命令を挿入する。図１３の文１４０がこれである。すなわち、最初の外側ループ繰り返し時における、最初の内側ループの最初に必要となるデータは、ここでプリフェッチをかける。しかし、このプリフェッチは、他の処理とオーバラップができないので、主記憶ペナルティが隠蔽できない部分である。図６のｊ＝１，ｉ＝１，２，３，４，５のプリフェッチ処理に相当する処理である。

【００３９】

【発明の効果】本発明のコード生成方式により、多重ループにおいて、アクセスするデータ領域がキャッシュ容量を超える場合も、内側ループ処理を２つのループに分け、前半ループに必要なデータは、１つ前の外側ループの繰り返し時の後半ループでデータのプリフェッチを開始し、後半ループで必要なデータは、前半ループでデータのプリフェッチを開始するため、主記憶ペナルティを隠蔽することができる。

【００４０】例えば従来の手法では、先に述べたように、外側ループ長Ｍに依存せず、内側ループ長Ｎが１００の場合、全体の時間の内４０％がデータ待ちの状態，内側ループ長Ｎが５０の場合は、全体の時間の内５６％がデータ待ちの状態であった。これに対して本発明の実＊

＊施によれば、最内側ループ長が９（＝ceil(７５／９)）以上であれば、外側ループの１回目の処理を除き、２回目以降の処理では、完全に主記憶ペナルティを隠蔽できる。即ち、内側ループ長が１００，５０のケースとも、外側ループ長が非常に大きければ、データ待ちの状態は発生しない。外側ループ長が５０の場合、内側ループ長が１００，５０の場合は、全体の時間の内、１.３％，２.６％がデータ待ちの状態で、外側ループ長が１０の場合、内側ループ長が１００，５０の場合は、全体の時間の内、６.２５％，１１.８％がデータ待ちの状態と改善することができる。

【図面の簡単な説明】

【図１】コンパイラ全体の構成図。

【図２】ソースプログラム例を示す図。

【図３】従来法による、内側ループ展開後のコード例を示す図。

【図４】従来法による多重ループにおけるプリフェッチ命令を使用したコード例を示す図。

【図５】従来法による実行の様子を示した説明図。

【図６】本発明による実行の様子を示した説明図。

【図７】本発明による多重ループの外側ループの展開方式の処理フロー図。

【図８】本発明による外側ループ展開後のコード例を示す図。

【図９】本発明による多重ループにおけるプリフェッチコード生成処理のフロー図。

【図１０】本発明による多重ループにおけるプリフェッチコード生成処理のフロー図。

【図１１】本発明によるループ分割後のコード例を示す図。

【図１２】本発明によるループ分割後の機械語に近いコード例を示す図。

【図１３】本発明による多重ループにおけるプリフェッチ命令を使用したコード例を示す図。

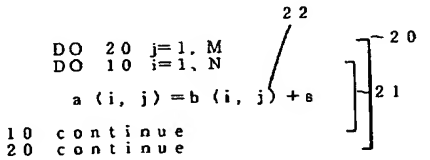
【図１４】本発明による多重ループにおけるプリフェッチ命令を使用したコード例を示す図。

【符号の説明】

１…ソースプログラム、２…構文解析部、３…中間語、４…ループ構造変換部、５…中間語、６…最適化部、７…中間語、８…コード生成部、９…オブジェクトコード。

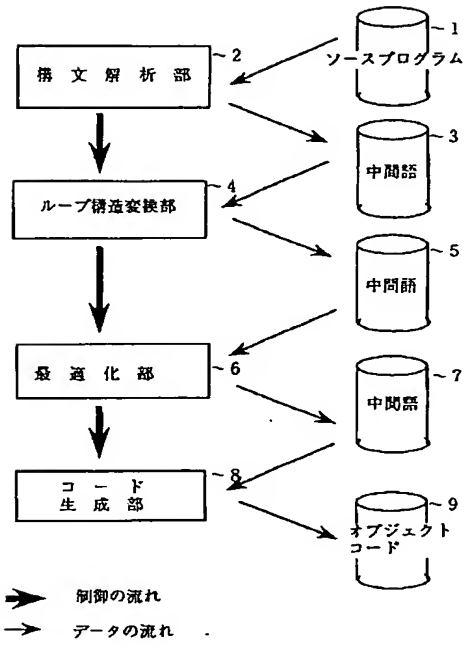
【図２】

図2



【図１】

図1



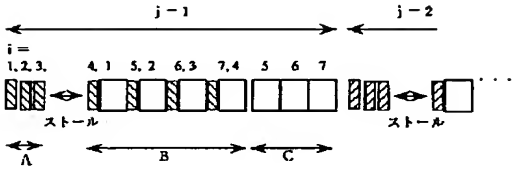
【図３】

図3

```
                                          24
    DO  20  j=1, M                        ┐
                                          25
      DO  10  i=1, N-3, 4                 ┐
        a (i, j) =b (i, j) +s             │
        a (i+1, j) =b (i+1, j) +s         │
        a (i+2, j) =b (i+2, j) +s         │
        a (i+3, j) =b (i+3, j) +s         │
10    continue                            ┘
      DO  11  i=1+ (N/4) *4, N, 4         ┐
        a (i, j) =b (i, j) +s             │
11    continue                            ┘
20  continue                              26 ┘
```

【図５】

図5



【図４】

図4

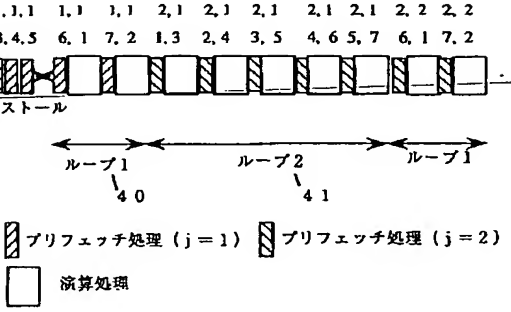
```
    DO  20  j=1, M
                                          30
      DO  1  i=1, min (N, 4*15), 4        ┐ A
        prefetch  b (i, j)                ┘
1     continue

                                          31
      DO  10  i=1, N-3-4*15, 4            ┐
        prefetch  b (i+4*15, j)           │
        a (i, j) =b (i, j) +s             │ B
        a (i+1, j) =b (i+1, j) +s         │
        a (i+2, j) =b (i+2, j) +s         │
        a (i+3, j) =b (i+3, j) +s         │
10    continue                            ┘
                                          32
      DO  11  i=max (1,                   ┐
          ((N-4*15)/4) *4), N-3, 4        │
        a (i, j) =b (i, j) +s             │ C
        a (i+1, j) =b (i+1, j) +s         │
        a (i+2, j) =b (i+2, j) +s         │
        a (i+3, j) =b (i+3, j) +s         │
11    continue                            ┘

20  continue
```

【図６】

図6



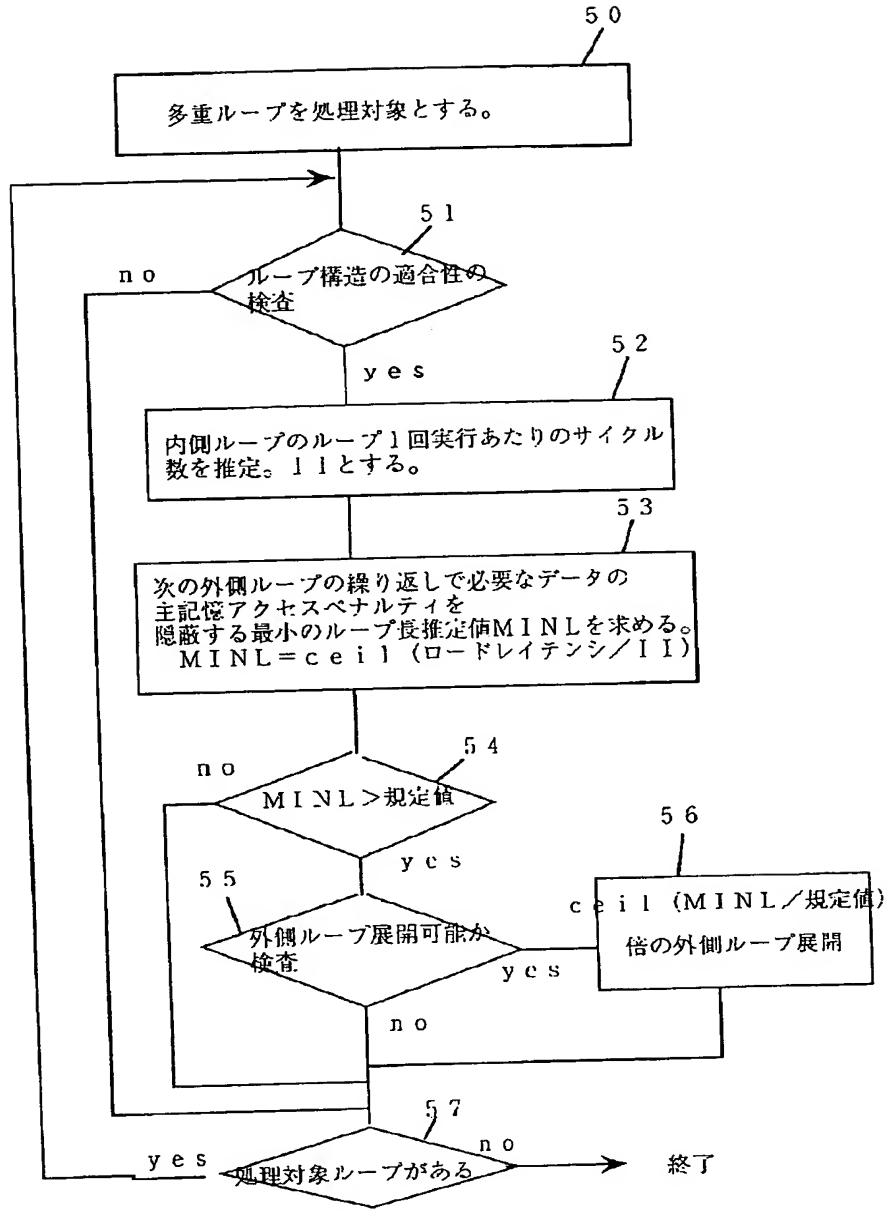
【図８】

図8

```
                                          60
                                          61
    DO  20  j=1, M-1, 2                   ┐
      DO  10  i=1, N                      │
        a (i, j) =b (i, j) +s             │
        a (i, j+1) =b (i, j+1) +s         │
10    continue                            ┘
20  continue
                                          62
    if (mod (M, 2) .ne. 0) then           ┐
      DO  11  i=1, N                      │
        a (i, M) =b (i, M) +s             ┘
11    continue
    endif
```

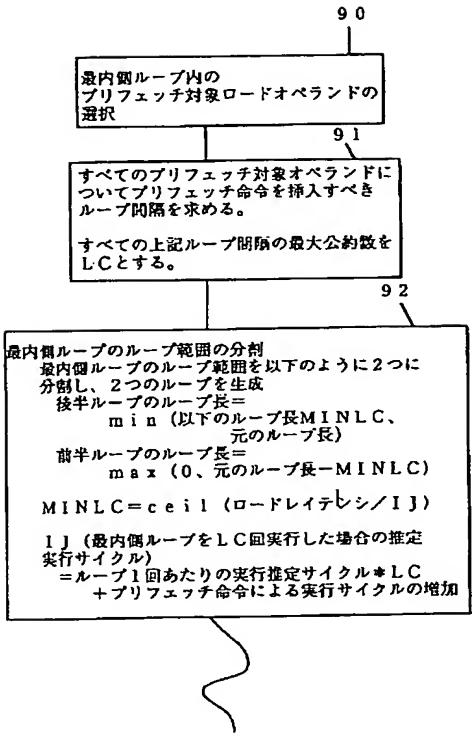
【図７】

図７

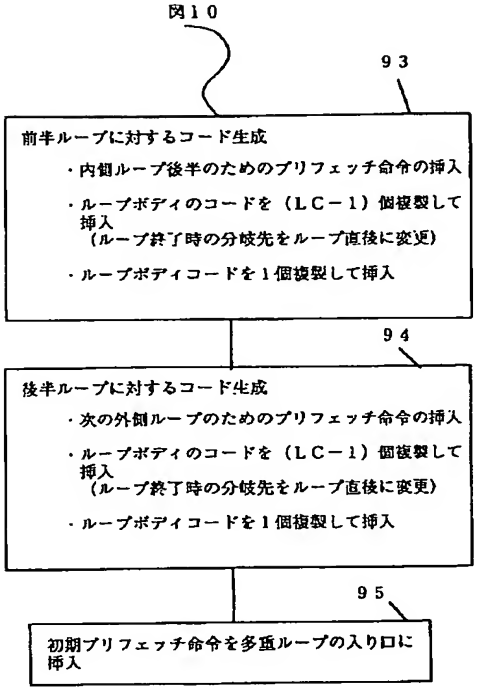

５０
多重ループを処理対象とする。
５１
ループ構造の適合性の検査
ｎｏ
ｙｅｓ
５２
内側ループのループ１回実行あたりのサイクル数を推定。ＩＩとする。
５３
次の外側ループの繰り返しで必要なデータの主記憶アクセスペナルティを隠蔽する最小のループ長推定値ＭＩＮＬを求める。
ＭＩＮＬ＝ｃｅｉｌ（ロードレイテンシ／ＩＩ）
５４
ＭＩＮＬ＞規定値
ｎｏ
ｙｅｓ
５５
外側ループ展開可能か検査
ｙｅｓ
ｎｏ
５６
ｃｅｉｌ（ＭＩＮＬ／規定値）倍の外側ループ展開
５７
処理対象ループがある
ｙｅｓ
ｎｏ
終了

【図９】

図９



【図１０】



【図１１】

図１１

```
      DO  20  j=1, M-1, 2                    65
        DO  10  i=1, N-ceil (75/9)
          a (i, j) =b (i, j) +s
          a (i, j+1) =b (i, j+1) +s
10      continue
        DO  11  i=                           66
            max (1,
            N-ceil (75/9) +1), N
          a (i, j) =b (i, j) +s
          a (i, j+1) =b (i, j+1) +s
11      continue

20    continue
```

【図１２】

図１２

```
      DO  20  j=1, M-1, 2

        i=1                                  ~70
        cnt=N-ceil (75/9) -1                 ~71
        if (cnt=0) goto  lab2                ~72
lab1:                                        ~73
          a (i, j) =b (i, j) +s              ~74
          a (i, j+1) =b (i, j+1) +s
          i=i+1                              ~75
          cnt=cnt-1                          ~76
          if (cnt .ne. 0) goto  lab1         ~77
lab2:                                        ~78


        i=max (1, N-ceil (75/9))             ~80
        cnt=N-max (1,                        ~81
            N-ceil (75/9) +1) +1
        if (cnt=0) goto  lab4                ~82
lab3:                                        ~83
          a (i, j) =b (i, j) +s              ~84
          a (i, j+1) =b (i, j+1) +s
          i=j+1                              ~85
          cnt=cnt-1                          ~86
          if (cnt .ne. 0) goto  lab3         ~87
lab4:                                        ~88


20    continue
```

【図13】

図13

```
      DO 1  i=1, min (N, ceil (75/9))
         prefetch  b (k, 1)
         prefetch  b (k, 2)                    ~140
         k=k+4
1     continue

      DO 20  j=1, M-1, 2

        i=1
        cnt=N-ceil (75/9) -1                   ~70
        k=1+ceil (75/9) *4                     ~71
        if (cnt=0) goto lab2                   ~100
lab1:                                          ~72
        prefetch  b (k, j)                     ~73
        prefetch  b (k, j+1)                   ~101
        k=k+4
        a (i, j) =b (i, j) +s                  ~102
        a (i, j+1) =b (i, j+1) +s              ~74
        i=i+1
        cnt=cnt-1                              ~75
        if (cnt = 0) goto lab2                 ~76
        a (i, j) =b (i, j) +s                  ~77
        a (i, j+1) =b (i, j+1) +s              ~103
        i=i+1
        cnt=cnt-1                              ~104
        if (cnt = 0) goto lab2                 ~105
        a (i, j) =b (i, j) +s                  ~106
        a (i, j+1) =b (i, j+1) +s              ~107
        i=i+1
        cnt=cnt-1                              ~108
        if (cnt = 0) goto lab2                 ~109
        a (i, j) =b (i, j) +s                  ~110
        a (i, j+1) =b (i, j+1) +s              ~111
        i=i+1
        cnt=cnt-1                              ~112
        if (cnt ≠ 0) goto lab1                 ~113
                                               ~114
lab2:
                                               ~78
```

【図14】

図14

```
        i=max (1, N-ceil (75/9))               ~80
        cnt=N-max (1,                          ~81
              N-ceil (75/9) +1) +1
        k=1
        if (cnt=0) goto lab4                   ~120
lab3:                                          ~82
        prefetch  b (k, j+2)                   ~83
        prefetch  b (k, j+3)                   ~121
        k=k+4
                                               ~122
        a (i, j) =b (i, j) +s
        a (i, j+1) =b (i, j+1) +s              ~84
        i=i+1
        cnt=cnt-1                              ~85
        if (cnt = 0) goto lab4                 ~86
                                               ~87
        a (i, j) =b (i, j) +s
        a (i, j+1) =b (i, j+1) +s              ~123
        i=i+1
        cnt=cnt-1                              ~124
        if (cnt = 0) goto lab4                 ~125
                                               ~126
        a (i, j) =b (i, j) +s
        a (i, j+1) =b (i, j+1) +s              ~127
        i=i+1
        cnt=cnt-1                              ~128
        if (cnt = 0) goto lab4                 ~129
                                               ~130
        a (i, j) =b (i, j) +s
        a (i, j+1) =b (i, j+1) +s              ~131
        i=i+1
        cnt=cnt-1                              ~132
        if (cnt ≠ 0) goto lab3                 ~133
                                               ~134
lab4:
                                               ~88

20    continue
```